CONTROLLER-LESS BOARD SWAP

NOTICE REGARDING COPYRIGHTED MATERIAL

FIELD OF THE INVENTION

This invention relates to live insertions and extractions of boards.

BACKGROUND OF THE INVENTION

Some high performance, "high availability" systems require that they suffer only minimal downtime (due to failures, maintenance, upgrades or system reconfigurations), and that often requires, in particular, insertion and extraction of boards while the system is active. For example, a satellite earth station system may dedicate to each of its many communications channels, its own board. There are spare boards on standby. When the subject board fails, it is desirable to direct the processing of the associated channel to a spare board and facilitate the replacement of the failed board, all without having to power down the system. This invention is directed to facilitating that replacement and related functions.

SUMMARY OF THE INVENTION

There is provided, in a system having an application and a plurality of boards, where the application is implemented by several software processes operating with the assistance of middleware between said boards and the application, and where said plurality of boards are adaptable to communicate pursuant to a standard that contemplates a controller dedicated therewith, a method for removing a subject board from the system while the application is running (a "hotswap"), comprising the steps of: a) connecting the boards with a communications channel in addition to that of the standard; b) preventing

any new transactions on the subject board; c) notifying relevant other boards; d) waiting for appropriate responses from relevant other boards; and e) waiting for subject board to quiesce; where steps b) to e) are effected using said additional communications channel.

BRIEF DESCRIPTION OF THE DRAWINGS

A better understanding of the present invention can be obtained when the following detailed description of the preferred embodiment is considered in conjunction with the following drawings, in which:

FIG. 1 is an example of a conventional CompactPCI® Hot Swap implementation (being Figure 39 in Appendix A of CompactPCI® Hot Swap Specification PICMG 2.1 R.20 (January 17, 2001); and

FIG. 2 is a circuit diagram that, according to the present invention, allows a board to satisfy both the CompactPCI® Hot Swap Specification and to detect the opening of its own handle.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Herein, a "device" typically has "microprocessor and related functionality", and a "board" is the physical implementation of a device (which typically includes a microprocessor but may implement "microprocessor functionality" with alternative means, like an FPGA or other Programmable Logic Device). The board has an interface with a human operator as follows - a handle (or similar electro-mechanical latch for the operator to manipulate to signal an intention to physically extract by generating an interrupt to the board's processor) and a blue LED (or similar indicator to indicate to the operator that it is safe to physically extract). The boards may communicate by a conventional bus means, such as CompactPCI but this invention does not require them to, since for some applications, the boards may simply be using the form factors of a conventional bus standard and not using the bus for communications.

The "live" insertion and/or extraction of a subject board (i.e. without powering down the system) is herein called generally "hotswap" (one word). When supported and implemented by a particular conventional bus standard (such as CompactPCI® Hot Swap

Specification PICMG 2.1 R2.0 January 17, 2001), the process is identified by the simplified name of the standard (i.e. "CompactPCI Hot Swap").

The term, "quiescent", and cognate terms, when applied to a board, refers to the state wherein all in-progress transactions have completed.

The system needs to react appropriately to an increase or decrease in available resources (e.g. by redistributing work among the available boards). To do so, it is important that notification of the change in configuration is distributed appropriately throughout the system. If there are direct communications between boards, it may be necessary for this direct communication to cease before the subject board is physically removed from the system.

In a CompactPCI system, extraction is a two-phase process. The first phase is the opening of the handle on the subject board, which generates a notification that its extraction is imminent. The system then has an opportunity to quiesce the subject board before lighting a blue LED on the subject board, to indicate that it is now safe for extraction. In a CompactPCI system, the notification takes the form of an interrupt to the System Host (which, as a central resource, provides configuration of the CompactPCI bus, and clocks and arbitration therefore, and plays a central role in "Dynamic Configuration", a process whereby a Hot Swap board is allocated system resources following insertion of the board). In systems where there is no traffic over the CompactPCI bus, handling this interrupt may be the only reason for a System Host to be present and thus it would be advantageous to operate without the System Host. Accordingly, it is advantageous if standard components recognized by conventional bus standards (e.g. for CompactPCI Hot Swap) are still present, allowing operation both with and without a System Host.

In an embedded system with no centralized control, entities which care about insertion or extraction events may be distributed on different processors (i.e. boards) throughout the system. Such entities may care about hotswap events in one of two ways: they may be interested in knowing about a hotswap event within some defined time of its occurrence, or they may need to know about an imminent hotswap event in order to take appropriate action before it occurs. In the former case, mere notification of completion of the event is sufficient. However the latter case demands an acknowledgement on the part

of the interested entity before the hotswap event occurs. The mechanism used for both cases should ideally provide complete decoupling of interested entities and sources of hotswap events and achieve an acceptable response time so that completion and notification of events is not unduly delayed.

This invention equips the boards (and processes therein and therewith) with additional means for communicating with each other, other than through the standard bus mechanism (e.g. independently of the CompactPCI bus). This additional communication means has two aspects, corresponding roughly to the lower and upper levels of the OSI reference model (e.g. cable and COBRA), explained next.

For the first aspect, to avoid inter-board communications proceeding exclusively through the bus for the boards, the boards are physically connected otherwise. This method of communication may be conventional (e.g. wires through which communications operates on Ethernet protocol, or wireless) or proprietary. Many current boards have built-in Ethernet ports and firmware such that only minimal steps are necessary to establish a non-bus communication channel (i.e. connecting all the boards by Ethernet cables) among them. Note that this provides the functional advantages of CompactPCI Hot Swap without the weaknesses thereof (i.e. there is no single points of failure as found in the case of a single System Host or a single CompactPCI backplane (because, in the exemplary preferred embodiment, the Ethernet communications among the boards eliminates such weaknesses).

The second aspect is advantageous for complex applications (although the degree of complexity of implementing the second aspect depends on the complexity of the application and performance requirements). The preferred embodiment uses CORBA (Common Object Request Broker Architecture) as an enabler for hotswap management among the boards. In particular, the CORBA Event Service or Notification Service provides a convenient method for communicating hotswap events through the system.

This second aspect is implemented by middleware residing between the hardware of the boards (including the subject board to be extracted) and the application. Middleware is implemented in the preferred embodiment, using three main components: (1) *quic*Comm, software library that supports inter-processor communications in a heterogeneous processing environment, from Spectrum Signal Processing Inc., (2)

VxWorks®, a real-time operating system for embedded architectures, from Wind River Systems, Inc., and (3) as mentioned above, CORBA, a conventional technology that allows objects within distributed object-oriented programs to invoke methods remotely. Related to the above main components and as part of middleware, is hardware-proximate interface software (such as board support libraries to map between *quic*Comm and hardware, and map between VxWorks and hardware), herein collectively called "Board Support".

*quic*Comm must respond to two types of hotswap events – insertions and extractions. In general, there are more rigorous timing requirements for extractions than for insertions. Conventionally, the notification of these events normally occurs in a "bottom-up" fashion, with a CompactPCI device driver detecting an interrupt, polling the CompactPCI bus to detect the subject device and proceeding from there. In an SDR (Software Defined Radio) system, it is desirable to allow for "top-down" notification, where a board may use a mechanism such as CORBA to inform other boards that it is being extracted, thus avoiding the requirement for a System Host to be present. Also, in an SDR system, there may be boards sending data over the switched fabric to the subject board to be extracted. Ideally, an application should be given a chance to stop this data flow before the blue LED is lighted.

A hotswap could affect *quic*Comm at all levels – from the highest level software objects (e.g. first call to *quic*Comm_Open() to open a system), down to device drivers. In essence, every software object may need to be aware that its corresponding hardware may be physically present or absent. That said, it may be possible in many cases to leave this knowledge at the lowest levels of the software hierarchy. If an attempt is made to send data to hardware that is not present, the lowest level will clearly be unable to fulfill this request and will return an appropriate error. As such attempts should be rare (because the application will be aware of the presence or absence of each board), the overhead involved in detecting such a problem later is generally acceptable.

Because of the need to allow the application sufficient time to communicate hotswap events to other boards and to wait for them to take appropriate action before lighting the blue LED of the subject board, most of the extraction process has to be performed within the context of the task that provides notification to *quic*Comm of the

event (as part of middleware's *quic*CommCompactPCIHot SwapLib). In other words, there needs to be an acknowledgement so that the subject board's middleware is satisfied.

To avoid "race conditions" between insertion and extraction events (i.e. when they occur in quick succession), it makes sense to do the same for the vast majority of extraction processing, too.

To ease application design, on system startup, all *quic*Comm objects will assume that the hardware they correspond to is not present until they find out otherwise. As boards are detected, simulated insertion events will be generated to the application. This allows the application to use the same code to deal with a board that is present when the system is first powered up and when the same board is later inserted.

There are four, fundamental, distinct events for the system to handle: Local Extraction, Remote Extraction, Local Insertions or Aborted Extraction, and Remote Insertions or Aborted Extractions. For each fundamental event, there is a process or sequence of steps to be performed as described below, where the Local Extractions are seen from the viewpoint of the subject board and the Remote Extractions are seen from the viewpoint of the other boards, in particular, and of the system, generally.

The essence of the extraction process for a subject board, is summarized in the following conceptual sub-processes:

1. Prevent any new transactions on the subject board
2. Notify relevant other boards
3. Wait for "appropriate responses" from relevant other boards
4. Wait for subject board to quiesce.

These sub-processes are not necessarily sequential. For example, sub-process 4 can be done before or concurrently with sub-processes 2 and 3. Also, an "appropriate response" may be an actual message from a relevant other board indicating to the subject board that it is appropriate, from that other board's point of view, for the subject board to be extracted; or an "appropriate response" may simply be waiting for a timeout whose duration is preset appropriately for the application, in general, or the boards, in particular.

The extraction process (and other relevant processes), implemented in more step-like fashion, is explained below. The parts of the system that need to know of a fundamental event, other than the subject board, are called herein for economy of

expression, "Remainder of the System". The examplary references are to specific *quic*Comm processes.

Local Extractions

1. Event reported by Board Support (e.g. *quic*CommCompactPCIHot SwapLib)
2. Prevent new transactions on all appropriate boards
3. Inform the application through middleware (e.g. *quic*Comm's HS_EventWait())
4. Wait for the application to respond through middleware (e.g. *quic*Comm HS_ExtractionHandledNotify(), or timeout)
5. Wait for appropriate boards to report that they are quiescent
6. Inform Remainder of the System that the subject board is no longer present
7. Return control to Board Support (e.g. *quic*Comm CompactPCIHot SwapLib) to light the blue LED

Remote Extractions

1. Application calls middleware (e.g. *quic*Comm HS_Quiesce()) to report the event
2. Prevent new transactions on all appropriate boards
3. Wait for appropriate boards to report that they are quiescent
4. Inform Remainder of the System that the subject board is no longer present
5. Return control to the application

Local Insertions or Aborted Extractions

1. Event reported by Board Support (e.g. *quic*CommCompactPCIHot SwapLib)
2. May need to extinguish the blue LED for an aborted extraction
3. Find appropriate software object(s)
4. Unquiesce appropriate boards, allowing them to be used
5. Inform Remainder of the System that the subject board is now present
6. Inform the application through middleware (e.g. *quic*Comm *c*HS_EventWait())

Remote Insertions or Aborted Extractions

1. Application calls middleware (e.g. *quic*Comm HS_InsertNotify()) to report the event
2. Unquiesce appropriate boards, allowing them to be used
3. Inform Remainder of the System that the subject board is now present
4. Return control to the application

FIG. 1 is an example of a conventional CompactPCI Hot Swap implementation, being page 140, Figure 39 in Appendix A of CompactPCI Hot Swap Specification PICMG 2.1 R.20 (January 17, 2001). FIG. 2 is a circuit diagram that, according to the present invention, allows a board to satisfy both the CompactPCI Hot Swap Specification (with examplary reference to FIG. 1) and to detect the opening of its own handle. Attached hereto, and incorporated as part of this application, is draft CompactPCI Hot Swap Specification PICMG 2.1 D0.91 (February 5, 1998), which is equivalent to CompactPCI Hot Swap Specification PICMG 2.1 R.20 (January 17, 2001) referred to herein for purposes of this invention (i.e. the differences between the draft and the standard are not relevant to this invention).

Ancillary to fundamental events are ones like system startup/shutdown, errors detected during board insertion. Also, Remote Extraction can be advantageously employed by an application that predicts the failures of boards and schedules their replacement before actual failures. The preceding are merely two examples where those skilled in the art can implement and use the present invention advantageously.

Also, the methods of the present invention can be employed advantageously where control of the fundamental events and processes resides remotely from the physical chassis where the boards are located and communications therebetween is accomplished by the additional means for communications (mentioned above, e.g. Ethernet cabling). Thus control of hotswap events (and processes and sub-processes) according to the present invention, can be centrally effected over a plurality of chassis of boards, thus negating or reducing the need for each chassis of boards to have their own "hotswap" control functionality.

Although the preferred embodiment has been described relative to the

CompactPCI standard, this invention is not restricted thereto. The invention is equally applicable, with obvious changes as applicable, to other standards for connecting (peripheral) boards via a bus. Examples include the PCI standard (PCI Local Bus Specification Rev. 2.2, December 18, 1998, and subsequent revisions), as well as the VME standard (Versa Module European - IEEE 1014-1987 standard) (whose "System Controller" is equivalent, for purposes of this invention, to CompactPCI's Hot Swap's "System Host").

Although in the preferred embodiment, middleware was implemented by *quic*Comm, VxWorks and CORBA, it is understood by those skilled in the art that other implementations are possible and are within the design choices of those skilled in the art.

Although in the preferred embodiment, an example of a "top-down" notification referred to an SDR system, any application whose software processes are complex, could advantageously be the subject of the present invention.

As mentioned above, the "middleware" depends on the complexity of the application and on performance requirements. Although the preferred embodiment uses a plurality of middleware software for a relatively complex application, it is conceivable that for a particular, simple application, the amount of middleware is so "thin" that it can be considered part of the application. Accordingly, the use of the term "middleware" herein should not be interpreted to preclude its inclusion in the term "application" in some situations. In some situations, there is no "discrete middleware" because the particular application, for its purposes, is sufficient to interact with the hardware without discrete middleware therebetween.

The CompactPCI Hot Swap standard provides for a "Hot Swap Controller", being a central device capable of exercising hardware connection control in effecting CompactPCI Hot Swaps. Whether this central device is implemented as (or considered) part of System Host or not for other purposes, this invention does not make a distinction and considers the Hot Swap Controller to included in the references to System Host as a matter of terminological convenience herein.

Although the preferred embodiment suggests Ethernet communications links among the boards, this invention does not restrict the type or topology of the links, although obviously, single-point-of-failure links are not preferred.

Details of *quic*Comm are publicly available at, for example, *quic*Comm API Programming Guide #DOC-500-00603 Rev. 1.00 Feb 2002 from Spectrum Signal Processing Inc.

Details of VxWorks are publicly available at VxWorks Programmer's Guide 5.4 Edition 1 25 Mar 99 Part # DOC-12629-ZD-01 from Wind River Systems, Inc.

Details of CORBA are publicly available in many publications including CORBA Notification Service Specification Version 1.0.1 from the OMG at http://www.omg.org/docs/formal/02-08-04.pdf and CORBA Event Service Specification Version 1.1 March 2001 from the OMG at http://www.omg.org/docs/formal/01-03-01.pdf.

Amplification of the CORBA implementation aspects of this invention is found below at the end of this specification, as pages 10a to 10g with drawings on pages 10h to 10j, which are incorporated herein as part of this specification.

Although the method and apparatus of the present invention has been described in connection with the preferred embodiment, it is not intended to be limited to the specific form set forth herein, but on the contrary, it is intended to cover such alternatives, modifications, and equivalents, as can be reasonably included within the spirit and scope of the invention as defined by the appended claims.

Distributed Hot Swap Management in an Embedded System

Abstract

Many modern embedded systems have a need for minimal downtime due to failures, maintenance or upgrades, which is addressed by permitting insertion and removal of boards while the system is active. Both CompactPCI and VME systems support live insertion and/or removal of boards, in CompactPCI, this is termed "Hot Swap". It is important that notification of such a change in configuration is communicated appropriately throughout the system so that it can react appropriately, perhaps by redistributing work between the available resources or shutting down any direct communication between boards before the boards involved are physically removed from the system. In a CompactPCI system, the notification takes the form of an interrupt to a System Host which takes the appropriate action. However, not only does the System Host constitute a single point of failure, but in many systems where there is no other traffic over the CompactPCI bus, handling this interrupt may be the only reason for the System Host to be present at all. Elimination of this centralized control function would therefore lead to a cheaper, more robust and optimal system architecture.

In an embedded multiprocessor system, entities which need to know about insertion or removal events may be distributed throughout the system. Such entities may care about Hot Swap events in one of two ways: They may be interested in knowing about a Hot Swap event within some defined time of it having occurred, or they may need to know about an impending Hot Swap event and take some action before it occurs. In the former case asynchronous notification of completion of the event is sufficient, however the latter case demands synchronous action on the part of the interested entity. The mechanism used for both cases should ideally provide complete decoupling of interested entities and sources of Hot Swap events and achieve an acceptable response time so that completion and notification of events is not unduly delayed.

This article discusses how a modified version of the Real-Time CORBA Notification Service may be used to meet these requirements.

CompactPCI Hot Swap

CompactPCI Hot Swap, as defined in the Hot Swap Specification [1], introduces the ability to insert or remove a Peripheral Board (any board other than the System Host) from a running system. Removal is a two-phase process. The first phase occurs when the lower latch on the board is opened, this generates an interrupt notifying the System Host that removal of the board is imminent. The Host then has an opportunity to quiesce the board before initiating the second phase by lighting a blue LED on the board, which indicates that it is now safe to fully extract the board. Hot Swap thus increases the availability of CompactPCI systems by allowing the removal and replacement of Peripheral Boards without the need to power down the entire chassis. Unfortunately, the System Host is still a single point of failure. If the System Host fails or needs to be upgraded, the entire chassis must still be powered down in order for it to be removed and replaced.

The Hot Swap specification imposes certain requirements on the software running on the System Host. In particular, on detecting that a Peripheral Board is about to be removed, the System Host is required to ensure that the board being extracted no longer accesses the PCI bus and that the board is not accessed from the PCI bus. This is necessary to ensure the integrity of signals on the bus as the board is removed. This requirement is reasonably easy to meet when all communication is between the Peripheral Board and the System Host, but significantly harder if there is direct communication between Peripheral Boards.

An Alternative Approach

In many CompactPCI systems, Peripheral Boards have a reasonable amount of intelligence. They may have fully functional CPUs and even run Operating Systems. For these boards, it is natural that they communicate directly with one another.
In systems with a large amount of data to process in a short period of time, the bandwidth of the CompactPCI bus may not be sufficient. This is exacerbated by the fact that the available bandwidth is shared between all the Peripheral Boards and the System Host.

In such a system, it may make sense for Peripheral Boards to communicate directly via some other medium, quite possibly in a point-to-point manner. This is the impetus behind the PICMG 2.16 [2], the forthcoming PICMG 2.18 [3] and other related standards, which introduce alternative communication links between Peripheral Boards. In systems such as these, the CompactPCI backplane may be little more than a source of power and a clock. A clock signal can easily be generated on the Peripheral Boards if it is not present on the backplane but is needed for a local PCI bus on the board. This leaves the non-redundant System Host with just one job – supporting Hot Swap of the Peripheral Boards.

Removing the System Host

If the work involved in handling Hot Swap events could be distributed to the Peripheral Boards, the System Host could be removed completely, eliminating the single point of failure. This turns out to be relatively simple to achieve.

In a CompactPCI Hot Swap system, Peripheral Boards have a micro-switch attached to the lower board latch. When the latch changes state between open and closed, the System Host is interrupted and can deal with the event. A Hot Swap Control and Status Register in configuration space of the device is used to communicate the details of the event. With a CPU on the Peripheral Board, it is relatively simple to implement similar functionality entirely on the board itself. Opening or closing the latch generates an interrupt to the CPU on the Peripheral Board. That CPU can query the hardware to determine exactly what happened and how to respond. It is even possible to implement a register that behaves exactly like a Hot Swap Control and Status Register, in which case the software that runs on the local CPU to handle Hot Swap events may be exactly the same software that usually runs on the System Host. This allows each Peripheral Board to independently detect that it is about to be removed.

In a system with no System Host, there can be no communication over the CompactPCI backplane, so there is nothing there that needs stopping. There may of course be other

communication links that need to be moved to a quiescent state. Surprisingly, there is a need to handle insertion as well as extraction. An insertion may look exactly like a power-up – the board moves from an un-powered state to one in which it has power. There is also the possibility that the latch was opened, causing the extraction processing to be performed, and then re-closed without actually extracting the board and causing it to power-cycle. This change in state needs to be handled in much the same way as when the board first gets powered up.
It is possible to design a board that detects whether there is a clock on the CompactPCI bus. If it detects such a clock, it should meet all the normal CompactPCI requirements. If it does not detect such a clock, it can assume that the CompactPCI backplane is not being used for communication and that there is no System Host present. In this case, the local CPU can take over the handling of Hot Swap management tasks as described previously.

Types of Interaction with Hot Swap Events
In an embedded system with no centralized control, entities which care about insertion or removal events (Hot Swap event consumers), may be distributed on different, heterogeneous processors throughout the system.

Such consumers may care about Hot Swap events in one of two ways:

1. They may merely need to react to a Hot Swap event after it has happened. For example different consumers may need to:
   - Initialize hardware that has been inserted.
   - Instantiate proxies for hardware that has been inserted.
   - Update the system state on a user interface.

   Although an asynchronous mechanism is sufficient, in fact desirable, for these types of notifications, it still needs to be efficient enough to achieve acceptable response times.

2. Alternatively they may need to be directly involved in an impending Hot Swap extraction event in order to perform some processing before the extraction actually happens. For example different consumers may need to:
   - Quiesce themselves and save their state if they are physically located on the hardware to be removed.
   - Shutdown communications if they have a direct connection to the hardware to be removed.
   - Reconfigure the application if they are acting in some type of system supervisory role.

   Clearly a synchronous mechanism is required to notify all directly involved entities in a timely manner and ensure that they have completed their processing before extraction is allowed to proceed.

General Requirements for Decentralized Hot Swap Event Notification
Thus in order to support decentralized Hot Swap in a heterogeneous computing environment, we require an event notification mechanism that:

- Provides both efficient synchronous and asynchronous notification services. (While not required, it is desirable that these services are accessed via a common interface)
- Supports diverse target processors.
- Supports location transparency so that events are distributed identically irrespective of whether they are local to the hardware in question, within the same chassis or somewhere else in an arbitrarily large system.
- Provides loose (i.e. determined at run-time) coupling between sources and consumers of Hot Swap events (since system configuration is by definition changing on the fly).
- Is standards-based in order to maximize re-use and portability.
- Offers redundancy to remove sensitivity to individual failures.

The Chosen Solution

A solution based on the Object Management Group's (OMG) CORBA Notification Service [4] is chosen for the following reasons:

- It is based upon the Common Object Request Broker Architecture (CORBA) specification [5] which is a well supported middleware industry standard which offers an architecture, language, transport and vendor independent communication in a location transparent manner.
- It provides an efficient asynchronous event distribution mechanism which offers advanced quality of service (QoS) and filtering properties (which were not present in its predecessor the CORBA Event Service). The QoS properties are used to control reliability, priority and timeliness of event distribution, while the filtering mechanism makes efficient implementation possible by suppressing unnecessary distribution of all events to all consumers.
- It provides the required loose coupling by means of a publish/subscribe model via which sources and consumers register independently with a well-known Hot Swap Event Channel and thus require no direct reference to each other.
- Provided the CORBA implementation supports it, the Hot Swap Event Channel may be implemented as a redundant object in accordance with the Fault Tolerant Specification [Chapter 23 of reference 5].

The Notification Service does not provide a synchronous event distribution but its interfaces and semantics may be extended to provide such a mechanism as discussed next.

Implementing a Hot Swap Event Channel

During system initialization a single Notification Service event channel is instantiated. This Hot Swap Event Channel registers itself at a well know location within the CORBA Naming Service.

Any consumer which wishes to be notified of any kind of Hot Swap event locates the Hot Swap Event Channel by means of the Naming Service and registers itself to receive Hot Swap Events (see Fig. 1). The filtering mechanism provided by the Notification Service allows a consumer to specify only the exact kinds (Insertion, Insertion_Error,

Pre_Extraction, Post_Extraction and Channel_Shutdown) and sources of Hot Swap events that it cares about. This eliminates the overhead of the service distributing and the consumer ignoring events which are of no interest to it.

For example, a consumer that needs to perform some processing before a particular piece of hardware is extracted, needs to register for the Pre_Extraction event from that particular source, while a consumer that needs to be notified of system configuration changes would register for the Insertion and Post_Extraction events from all sources. The Insertion_Error event is generated when hardware is inserted but is unusable for some reason (for example a self-test failure). Some consumers might register for all events. The consumer will be called back by the Hot Swap Event Channel ("Channel") when an event in which it is interested (i.e. matching its filter) occurs.

Figure_1_Hot_swap_event_channel.gif

Figure 1: Hot Swap Event Channel Initialization Sequence

Sources of Hot Swap events are typically service routines responding to the Hot Swap interrupt on processors residing on removable hardware, however the System Host in a CompactPCI chassis which contains only dumb peripherals may also be a source distributing events to a wider Super-System. It is also possible for a device (such as a system health monitor) in the system to inject Hot Swap events for boards on which it detects a failure. When a Hot Swap source has an event to distribute, it locates the Hot Swap Event Channel in the Naming Service and calls the Channel with the event.

Thus a push model for event distribution is employed where sources push events to the Channel, which in turn pushes the events to interested consumers. Structured events as defined by the Notification Service are used since they provide filtering support.

Asynchronous Event Distribution

Asynchronous distribution is used whenever a source has an Insertion, Insertion_Error, Post_Extraction or Channel_Shutdown to report. This is the normal "fire and forget" mode of operation of the Notification Service in which the push call from the source returns immediately, the Channel thereafter passes the event through the applicable filters and distributes it to all interested, registered consumers (see Fig. 2). In an efficient implementation, multiple threads are used to notify interested consumers simultaneously [6].

Figure_2_Asynchronous_event.gif

Figure 2: Asynchronous Event Distribution Sequence

Synchronous Event Distribution

In order to support synchronous participation in Pre_Extraction events, an extension to the Notification Service is needed to provide synchronous behaviour. The semantics of this extension otherwise match the standard Notification Service specification.

A synchronous event is generated when a source calls the push operation on a newly defined synchronous interface on the extended Notification Service. This event is distributed in exactly the same way as an asynchronous event, except that the source is called back when all consumers have completed their processing (or otherwise failed or timed-out) with an indication of the success or reason for failure of the total operation (see Fig. 3). This permits the source to wait for completion of the event distribution before continuing or alternatively implement its own timeout if no response is received.

Figure_3_Synchronous_pre_extraction.gif

Figure 3: Synchronous Pre_Extraction Event Distribution Sequence

Extraction Processing

Thus each extraction which is initiated, first generates a synchronous Pre_Extraction event notification to ensure that all required pre-extraction processing is complete, followed by an asynchronous Post_Extraction event notification prior to lighting the blue LED for extraction to complete.

Fault Tolerance

One of the stated advantages of eliminating the need for a System Host from a CompactPCI bus was the removal of a single point of failure from the system, however when using the extended Notification Service all events must pass through a Hot Swap Event Channel which must reside on a single processor which again becomes a single point of failure.

This problem is addressed by the Fault Tolerant CORBA specification by providing for distributed redundant replicas of CORBA server objects and mechanisms to switch between these replicas to deal with failures. This is completely transparent to users of the service. Therefore, if required, the Hot Swap Event Channel can be implemented as a Fault Tolerant object to provide additional system reliability without impacting Hot Swap event sources and consumers.

The Hot Swap Event Channel therefore meets all the above requirements for a decentralized Hot Swap control mechanism.

Summary

With the aid of appropriately configured middleware, it is possible to distribute Hot Swap control among intelligent computing devices in a CompactPCI chassis thereby eliminating the need to centralize this function on a System Host. Moreover events may be distributed throughout a Super-System consisting of multiple chassis both with and without System Hosts. This has both reliability and system management benefits.

An enhanced implementation of the CORBA Notification Service provides a suitable middleware platform to support distributed Hot Swap control.

References

[1] CompactPCI Hot Swap Specification PICMG 2.1 R2.0 January 17, 2001

[2] PICMG 2.16 IP Backplane for CompactPCI R1.0 September 5, 2001
[3] PICMG 2.18 Serial RapidIO over CompactPCI
[4] Object Management Group, Notification Service Specification, Version 1.0.1, OMG Document formal/2002-08-04
[5] Object Management Group, Common Object Request Broker Architecture (CORBA/IIOP), Version 3.0.2, OMG Document formal/2002-12-06
[6] Pradeep Gore, Douglas C. Schmidt, Carlos O'Ryan, and Ron Cytron, "Designing and Optimizing a Scalable CORBA Notification Service", Proceedings of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001), Snowbird, Utah, June 18, 2001.